AUTOMATIC TRANSFORMATION OF IRREDUCIBLE REPRESENTATIONS FOR EFFICIENT CONTRACTION OF TENSORS WITH CYCLIC GROUP SYMMETRY

YANG GAO*, PHILLIP HELMS†, GARNET KIN-LIC CHAN‡, AND EDGAR SOLOMONIK§

Abstract. Tensor contractions are ubiquitous in computational chemistry and physics, where tensors generally represent states or operators and contractions are transformations. In this context, the states and operators often preserve physical conservation laws, which are manifested as group symmetries in the tensors. These group symmetries imply that each tensor has block sparsity and can be stored in a reduced form. For nontrivial contractions, the memory footprint and cost are lowered, respectively, by a linear and a quadratic factor in the number of symmetry sectors. State-of-the-art tensor contraction software libraries exploit this opportunity by iterating over blocks or using general block-sparse tensor representations. Both approaches entail overhead in performance and code complexity. With intuition aided by tensor diagrams, we present a technique, irreducible representation alignment, which enables efficient handling of Abelian group symmetries via only dense tensors, by using contraction-specific reduced forms. This technique yields a general algorithm for arbitrary group symmetric contractions, which we implement in Python and apply to a variety of representative contractions from quantum chemistry and tensor network methods. As a consequence of relying on only dense tensor contractions, we can easily make use of efficient batched matrix multiplication via Intel's MKL and distributed tensor contraction via the Cyclops library, achieving good efficiency and parallel scalability on up to 4096 Knights Landing cores of a supercomputer.

1. Introduction. Tensor contractions are computational primitives found in many areas of science, mathematics, and engineering. In this work, we describe how to accelerate tensor contractions involving block sparse tensors whose structure is induced by a cyclic group symmetry or a product of cyclic group symmetries. Tensors of this kind arise frequently in many applications, for example, in quantum simulations of many-body systems. By introducing a remapping of the tensor contraction, we show how such block sparse tensor operations can be expressed almost fully in terms of dense tensor operations. This approach enables effective parallelization and makes it easier to achieve peak performance by avoiding the complications of managing block sparsity. We illustrate the performance and scalability of our approach by numerical examples drawn from the contractions used in tensor network algorithms and coupled cluster theory, two widely used methods of quantum simulation.

A tensor $\mathcal{T}$ is defined by a set of real or complex numbers indexed by tuples of integers (indices) $i, j, k, l, \ldots$, where the indices take integer values $i \in 1 \ldots D_i, j \in 0 \ldots D_j, \ldots$ etc., and a single tensor element is denoted $t_{ijkl\ldots}$. We refer to the number of indices of the tensor as its *order* and the sizes of their ranges as its *dimensions* ($D_i \times D_j \times \cdots$). We will call the set of indices *modes* of the tensor. Tensor contractions are represented by a sum over indices of two tensors. In the case of matrices and vectors, the only possible contractions correspond to matrix and vector products. For higher order tensors, there are more possibilities, and an example of a contraction of two order 4 tensors is

$$w_{abij} = \sum_{k,l} u_{abkl} v_{klij}. \tag{1.1}$$

To illustrate the structure of the contraction, it is convenient to employ a graphical notation where a tensor is a vertex with each incident line representing a mode, and contracted modes are represented by lines joining vertices, as shown in Figure 1. Tensor contractions can be reduced to matrix multiplication (or a simpler matrix/vector operation) after appropriate transposition of the data to interchange the order of modes.

*Division of Engineering and Applied Science, California Institute of Technology (ygao@caltech.edu)
†Division of Chemistry and Chemical Engineering, California Institute of Technology (phelms@caltech.edu)
‡Division of Chemistry and Chemical Engineering, California Institute of Technology (garnetc@caltech.edu)
§Department of Computer Science, University of Illinois at Urbana-Champaign (solomonik@cs.illinois.edu)

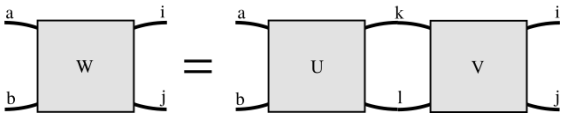

FIGURE 1. *Representation of the contraction in Eq. (1.1). Each tensor is represented by a vertex and each mode by a line; the lines joining the vertices are contracted over. Labelling the modes indexes into the tensor.*

In many applications, the tensors of interest contain symmetries, which allow each tensor to be stored in a compressed form, referred to as its *reduced form*. A special type of sparsity that often appears is one that is associated with a cyclic group structure defined on the indices, e.g.

$$t_{ijk\ldots} = 0 \quad \text{if} \quad \lfloor i/G_1 \rfloor + \lfloor j/G_2 \rfloor + \lfloor k/G_3 \rfloor + \cdots \neq 0 \pmod{G}. \tag{1.2}$$

For a matrix, such sparsity would lead to a blocked matrix where each block is the same size, $G_1 \times G_2$. The blocks of an order 3 tensor would similarly all have the same dimensions, $G_1 \times G_2 \times G_3$. We refer to such tensors as tensors with *cyclic group symmetry*, or cyclic group tensors for short. In quantum physics and chemistry these symmetries arise due to physical symmetries and conservation laws in the system. For example, when quantum states are classified by irreducible representations (irreps) of such symmetries, their numerical representation in terms of tensors can be chosen to carry this block structure. In some applications, the block sizes are non-uniform, but this can be accommodated in a cyclic group tensor by padding blocks with zeros to a fixed size. With this assumption, the block symmetry can then be expressed via an unfolding of $\mathcal{T}$, so that new symmetry modes yield indices that iterate over blocks,

$$t_{iI,jJ,kK\ldots} = 0 \quad \text{if} \quad I + J + K \cdots \neq 0 \pmod{G},$$

where we use the convention that the uppercase indices are the symmetry modes and the lowercase letters index into the symmetry blocks (physical modes).

Given a number of symmetry sectors G (as in (1.2)), cyclic group symmetry can reduce tensor contraction cost by a factor of G for some simple contractions and G^2 for most contractions of interest (any contraction with a cost that is superlinear in input/output size). State-of-the-art sequential and parallel libraries for handling cyclic group symmetry, both in specific physical applications and in domain-agnostic settings, typically iterate over appropriate blocks within a block-sparse tensor format [1, 4, 13, 16, 18–21, 28, 32, 34, 39]. The use of explicit looping (over possibly small blocks) makes it difficult to reach theoretical peak compute performance. Parallelization of block-wise contractions can be done manually or via specialized software [13, 16, 19–21, 28, 32, 34]. However, such parallelization is challenging in the distributed-memory setting, where block-wise multiplication might (depending on contraction and initial tensor data distribution) require communication/redistribution of tensor data. We introduce a general transformation of cyclic group symmetric tensors, *irreducible representation alignment*, which allows all contractions between such tensors to be transformed into a single large dense tensor contraction with optimal cost, in which the two input reduced forms as well as the output are indexed by a new auxiliary index. This transformation provides three advantages:

1. it avoids the need for data structures to handle block sparsity or scheduling over blocks,
2. it makes possible an efficient software abstraction to contract tensors with cyclic group symmetry,

3. it enables effective use of parallel libraries for dense tensor contraction and batched matrix multiplication.

The most closely related previous work to our approach that we are aware of is the *direct product decomposition (DPD)* [24,43], which similarly seeks an aligned representation of the two tensor operands. However, the unfolded structure of cyclic group tensors in Eq. (1) allows for a much simpler conversion to an aligned representation, both conceptually and in terms of implementation complexity. In particular, our approach can be implemented efficiently with existing dense tensor contraction primitives.

We study the efficacy of this new method for tensor contractions with cyclic group symmetry arising in physics and chemistry applications. In particular, we consider some of the costly contractions arising in tensor network methods for quantum many-body systems and in coupled cluster theory for electronic structure calculations. We develop a software library, *Symtensor*, that implements the irrep alignment algorithm and contraction. We demonstrate that across a variety of tensor contractions, the library achieves orders of magnitude improvements in parallel performance and a matching sequential performance relative to the manual loop-over-blocks approach. The resulting algorithm may also be easily and automatically parallelized for distributed-memory architectures. Using the Cyclops Tensor Framework (CTF) library [40] as the contraction backend to Symtensor, we demonstrate good strong and weak scalability with up to at least 4096 Knights Landing cores of the Stampede2 supercomputer.

2. Applications of Cyclic Group Symmetric Tensors. Cyclic group tensors arise in a variety of settings in computational quantum chemistry and physics methods. This is because both the quantum Hamiltonian and its eigenstates transform as irreps of the physical symmetry operators, for example, those associated with particle number, spin, and point group and lattice symmetries [6, 22, 44]. Many of these symmetries are associated with finite cyclic groups (e.g. finite Abelian point groups or finite lattice groups) or infinite cyclic groups (e.g. particle number symmetry and magnetic spin symmetry). Those associated with infinite cyclic groups give rise to conservation laws via Noether's theorem.

In this section, we summarize some of the quantum chemistry and physics applications of cyclic group tensors, and related developments to handle cyclic group symmetry within them. The performance evaluation of our proposed algorithms later considers some of the computationally expensive tensor contractions arising in these applications.

2.1. Electronic Structure. In electronic structure calculations in chemistry and physics, the most complicated part of the discretized Hamiltonian can be written as an order 4 tensor, often called the *two-electron integral tensor*. Similarly, the many-body eigenstates and wavefunctions are represented by tensors. One common choice of approximate wavefunction, arising in so-called correlated wavefunction methods such as configuration interaction (CI) [45] and coupled cluster (CC) methods at the doubles level of approximation [3, 7, 12, 46, 47], represents an important part of the wavefunction in terms of another order 4 tensor, the *doubles amplitude tensor*. Higher order tensors arise in approximations that target higher accuracy [17, 29, 35, 42].

Both the two-electron integral tensor and the amplitude tensors possess the cyclic group symmetries associated with the physical symmetries of the system. In a molecule, the largest symmetry group for which cost reductions can be achieved is usually the *point group symmetry*, e.g. associated with a rotational group C_{nv} or a product group such as D_{2h}. In crystalline materials, crystal translational symmetry is equivalent to a cyclic symmetry along each lattice direction. Typically the cyclic groups (number of k-points) are chosen to be as large as computationally feasible. Thus the savings arising from efficient use of translational symmetry are particularly important in materials simulations [11, 14, 15, 25, 27].

Many quantum chemistry implementations leverage cyclic group symmetries within var-

ious approximate quantum methods [25, 38, 43]. In addition, a number of state-of-the-art tensor software efforts aimed at quantum chemistry applications handle cyclic group symmetry in a general way via block sparse tensor formats [13, 16, 19, 21, 28, 32, 34]. As described in Section 3.1, our proposed algorithm achieves the same computational improvement via transforming the cyclic group tensor representation, while maintaining a global view of the problem as a dense tensor contraction, as opposed to a series of block-wise operations or a contraction of block-sparse tensors.

2.2. Tensor Network Simulation. Tensor network (TN) methods represent a quantum state, and the associated operators acting on it, as a contraction of tensors on the sites of a lattice. The contraction of the site tensors yields a full state or operator tensor with a number of modes equal to the size of the lattice (i.e. an exponentially large number of entries in the size of the lattice). The simplest such tensor network is the *matrix product state (MPS)* [9,51] (also referred to as a *tensor train* [31]), which is a 1D tensor network. It is widely used as part of the *density matrix renormalization group (DMRG)* method [51, 52]. DMRG uses an alternating optimization along the 1D tensor network to arrive at representations of ground or excited eigenstates of a Hamiltonian, which is often itself represented as a tensor train (*matrix product operator (MPO)* [5, 8, 49]).

Some common cyclic group symmetries in this context, known as Abelian quantum numbers, are those associated with the magnetic spin quantum number and particle number symmetries (both $U(1)$ symmetries) as well as fermionic supersymmetry (Z_2). Tensor networks that go beyond a 1D contraction structure have substantially more expressive power [48, 50, 53]. The *projected entangled pair states (PEPS)* are one such class defined in arbitrary dimensions (reducing to MPS in 1D) and are believed to provide a compact representation of the Hamiltonian eigenstates of physical interest in higher dimensions [48]. These more complicated tensor networks exhibit the same cyclic group symmetries as found in matrix product states. Because the symmetry groups arising in tensor network simulations can be arbitrary large (e.g. $U(1)$) their efficient use can lead to orders of magnitude improvement in memory footprint and time to solution [4, 26, 36, 52]. Similarly to the situation in quantum chemistry, tensor contraction libraries have been developed to handle these symmetries where the most common strategy is to track and loop over block-sparse representations [1,18,30,36]. These contractions and block-sparse representations can similarly be replaced by the cyclic group tensor representation and irrep alignment algorithms of this work. In Section 5.2, we study two representative contractions: one arising in an MPS simulation using the DMRG algorithm, and one in the contraction of a PEPS network.

3. Irreducible Representation Alignment Algorithm. We now describe our proposed approach. We first describe the algorithm on an example contraction and provide intuition for correctness based on conservation of flow in a tensor diagram graph. These arguments are analogous to the conservation arguments used in computations with Feynman diagrams (e.g. momentum and energy conservation) [10] or with quantum numbers in tensor networks [37], although the notation we use is slightly different. We then give a complete algebraic derivation of all steps, accompanied by more detailed tensor diagrams to explain each step.

3.1. Example of the Algorithm. We consider a contraction of order 4 tensors $\mathcal{U}$ and $\mathcal{V}$ into a new order 4 tensor $\mathcal{W}$, where all tensors have cyclic group symmetry. We can express this cyclic group symmetric contraction as a contraction of tensors of order 8, by separating indices into intra-block (lower-case) indices and symmetry (upper-case) indices, so

$$w_{aA,bB,iI,jJ} = \sum_{k,K,l,L} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}.$$

Algorithm 3.1 Loop nest to perform group symmetric contraction $w_{aA,bB,iI,jJ} = \sum_{k,K,l,L} u_{aA,bB,kK,lL} v_{kK,lL,iI,jJ}$.

for $A = 1, \ldots, G$ **do**
  for $B = 1, \ldots, G$ **do**
    for $I = 1, \ldots, G$ **do**
      $J = A + B - I \bmod G$
      for $K = 1, \ldots, G$ **do**
        $L = -A - B - K \bmod G$
        $\forall a, b, i, j, \quad w_{aA,bB,iI,jJ} = \sum_{k,l} w_{aA,bB,iI,jJ} + u_{aA,bB,kK,lL,} v_{kK,lL,iI,jJ}$
      end for
    end for
  end for
end for

Here and later we use commas to separate index groups for readability, each character is a distinct index. The input and output tensors have a group symmetry defined by a cyclic group, so that

$$\begin{aligned} w_{aA,bB,iI,jJ} &\neq 0 \text{ if } A + B - I - J \equiv 0 \pmod{G}, \\ u_{aA,bB,kK,lL} &\neq 0 \text{ if } A + B + K + L \equiv 0 \pmod{G}, \\ v_{kK,lL,iI,jJ} &\neq 0 \text{ if } K + L - I - J \equiv 0 \pmod{G}. \end{aligned}$$

Ignoring the symmetry, this tensor contraction would have cost $O(n^6 G^6)$, where n is the dimension of each symmetry sector, but with the use of group symmetry this is reduced to $O(n^6 G^4)$.

Existing approaches implement the contraction with $O(G^4)$ block-wise operations via a manual loop nest, as shown in Algorithm 3.1. The order 3 reduced forms $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{U}}$, and $\boldsymbol{\mathcal{V}}$ can be written as dense tensors and their elements may be accessed via the appropriate 3 indices to perform the multiply and accumulate operation in Algorithm 3.1. Reduced forms provide an implicit representation of one of the tensor modes (see Figure 2).

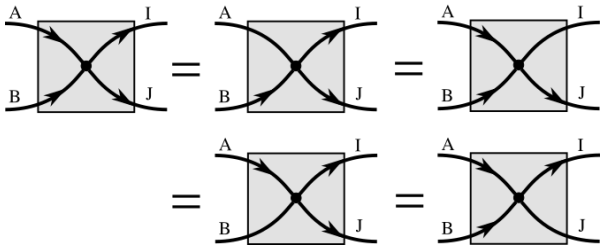

FIGURE 2. *Tensor diagrams of standard forms of reduced tensors (legs with arrows are indices/modes of the reduced tensor, legs without arrows are represented implicitly via an irrep map). The sum of the 4 arrows at the vertex (dot) satisfies* $A + B = I + J \pmod{G}$*, making one of the arrows redundant.*

However, the indirection needed to compute L and J within the innermost loops prevents expression of the contraction in terms of standard library operations for contraction of dense tensors. Figure 3 illustrates that standard reduced forms, where some choice of 3 symmetry indices of the original tensor is represented in each reduced form, cannot simply be contracted to obtain a reduced form as a result. This problem has motivated the use of sparse or block sparse representations of the tensors, which permit a work efficient approach

for arbitrary group symmetries [16]. However, the need to parallelize general block-wise tensor contraction operations creates a significant software-engineering challenge and computational overhead for tensor contraction libraries.

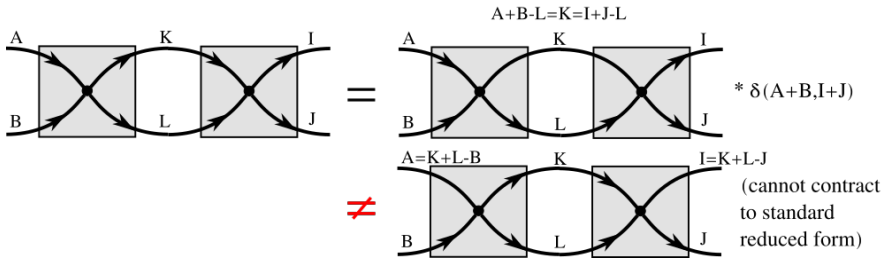

FIGURE 3. *These two tensor diagram equations aim to illustrate why certain reduced forms cannot be contracted directly. With the reduced forms chosen in the top case, the two symmetry relations of the implicitly stored index K mean that the contraction must be multiplied by a factor of $\delta(A+B, I+J)$. Without this, the contraction would perform too much computation. In the second case, the reduced forms cannot be contracted to produce a valid standard reduced form for the output (one needs 3 of the uncontracted indices to be represented / marked with arrows).*

The main idea in the irreducible representation alignment algorithm is to first transform (reindex) the tensors using an auxiliary index which subsequently allows a dense tensor contraction to be performed without the need for any indirection or symmetry rules. In the above contraction, we define the auxiliary index as $Q \equiv -I - J \equiv A + B \equiv K + L \pmod{G}$ and thus obtain a new reduced form for each tensor,

$$\begin{aligned}
\hat{w}_{aA,b,i,jJ,Q} &= w_{aA,b,Q-A \bmod G,i,-J-Q \bmod G,jJ}, \\
\hat{u}_{aA,b,k,lL,Q} &= u_{aA,b,Q-A \bmod G,k,Q-L \bmod G,lL}, \\
\hat{v}_{k,lL,i,jJ,Q} &= v_{k,Q-L \bmod G,lL,i,-J-Q \bmod G,jJ}.
\end{aligned}$$

This reduced form is displayed in Figure 4.

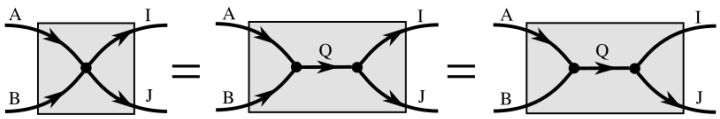

FIGURE 4. *The symmetry aligned reduced form is defined by introducing the Q index. Each of the two vertices defines a conservation rule: $A + B = Q \pmod{G}$ and $Q = I + J \pmod{G}$, allowing two of the arrows to be removed in the 3rd diagram, i.e. to be represented implicitly as opposed to being part of the reduced form.*

The Q index is chosen so that it can serve as part of the reduced forms of each of $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$. An intuition for why this alignment is possible is given via tensor diagrams in Figure 5. The new auxiliary indices (P and Q) of the two contracted tensors satisfy a conservation law $P = Q$, and so can be reduced to a single index.

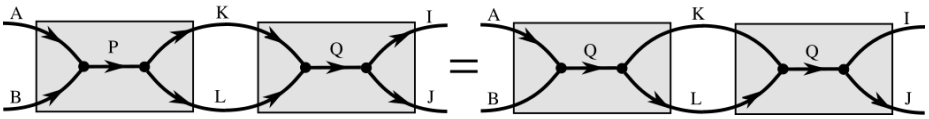

FIGURE 5. *By defining conservation laws on the the vertices, we see that $P = K + L \pmod{G}$ and $K + L = Q \pmod{G}$. Consequently, the only non-zero contributions to the contraction must have $P = Q$.*

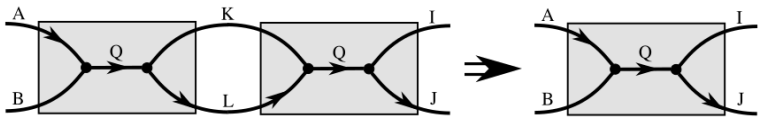

FIGURE 6. *The reduced forms may be contracted efficiently to produce the output reduced form. Ignoring intra-block indices, the resulting contraction may be performed with the* `einsum` *operation* `W=einsum("AQL,LQJ->AQJ",U,V)`.

As shown in Figure 6, given the aligned reduced forms of the two operands, we can contract them directly to obtain a reduced form for the output that also has the internal symmetry index Q. Specifically, it suffices to perform the dense tensor contraction,

$$\hat{w}_{aA,b,i,jJ,Q} = \sum_{L,k,l} \hat{u}_{aA,b,k,lL,Q} \hat{v}_{k,lL,i,jJ,Q}.$$

This contraction can be expressed as a single `einsum` operation (available via NumPy, CTF, etc.) and can be done via a batched matrix multiplication (available in Intel's MKL). Once $\hat{\mathcal{W}}$ is obtained in this reduced form, it can be remapped to any other desired reduced form.

3.2. General Algorithm Derivation. We now provide a formal derivation of the algorithm for contractions of tensors of arbitrary order, with cyclic group symmetries described by arbitrary coefficients. Algorithm 3.2 provides a description of each computational step, and associated costs. We represent an order N complex tensor with cyclic group symmetry as in (1.2) as an order $2N$ tensor, $\mathcal{T} \in \mathbb{C}^{n_1 \times H_1 \times \cdots \times n_N \times H_N}$ satisfying, for some implicit mode index $j \in \{1 \dots N\}$, modulus remainder $Z \in \{1 \dots G\}$, and coefficients $c_1 \dots c_N$ with $c_i = G/H_i$ or $c_i = -G/H_i$,

(3.1)
$$t_{i_1 I_1 \dots i_N I_N} = \begin{cases} r^{(T)}_{i_1 I_1 \dots i_{j-1} I_{j-1} i_j i_{j+1} I_{j+1} \dots i_N I_N} & : c_1 I_1 + \cdots + c_N I_N \equiv Z \pmod{G} \\ 0 & : \text{otherwise,} \end{cases}$$

where the order $2N-1$ tensor $\mathcal{R}^{(T)}$ is the *reduced form* of the cyclic group tensor $\mathcal{T}$. Any cyclic group symmetry may be more generally expressed using an irrep map, represented as a tensor with binary values, $\mathcal{M}^{(T)} \in \{0,1\}^{H_1 \times \cdots \times H_N}$ as

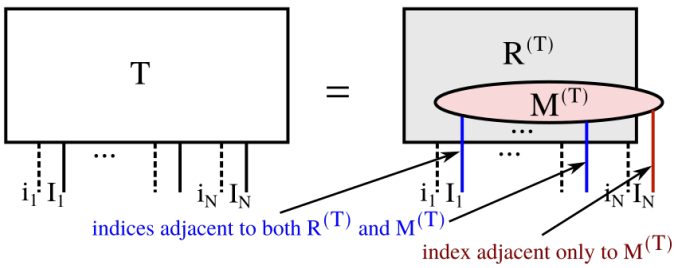

FIGURE 7. *Group symmetric tensor expressed as a product of a reduced form and an irrep map. Intra-block indices are denoted with dashed lines and symmetry indices are denoted with solid lines.*

$$t_{i_1 I_1 \dots i_N I_N} = r^{(T)}_{i_1 I_1 \dots i_{j-1} I_{j-1} i_j i_{j+1} I_{j+1} \dots i_N I_N} m^{(T)}_{I_1 \dots I_N}.$$

The tensor diagram form of this equation with $j = N$ is given in Figure 7.

Specifically, the irrep map is defined by

$$m^{(T)}_{I_1 \dots I_N} = \begin{cases} 1 & : c_1 I_1 + \cdots + c_N I_N \equiv Z \pmod{G} \\ 0 & : \text{otherwise.} \end{cases}$$

An irrep map can be decomposed by any tree tensor network, where every bond dimension has rank n, which follows from the properties of group multiplication in a cyclic group, and is explicitly shown via the following irrep map decomposition lemma. The equation in the lemma is displayed in terms of tensor diagrams in Figure 8.

LEMMA 3.1 (Irrep Map Decomposition). *Any irrep map* $\boldsymbol{\mathcal{M}} \in \{0,1\}^{H_1 \times \cdots \times H_N}$ *may be written as a contraction of two irrep maps* $\boldsymbol{\mathcal{M}}^{(A)} \in \{0,1\}^{H_1 \times \cdots \times H_k \times G}$ *and* $\boldsymbol{\mathcal{M}}^{(B)} \in \{0,1\}^{H_{k+1} \times \cdots \times H_N \times G}$,

$$m_{I_1 \dots I_N} = \sum_{J=0}^{G-1} m^{(A)}_{I_1 \dots I_k J} m^{(B)}_{I_{k+1} \dots I_N J}.$$

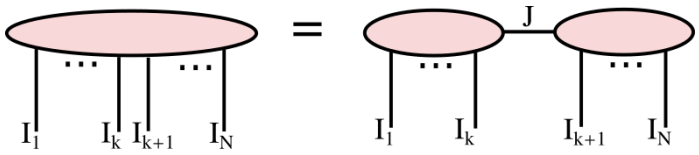

FIGURE 8. *An irrep map tensor may be decomposed with low rank (with a bond index of dimension equal to the size of the group, G, no matter the order of the irrep map tensor) as a contraction of two irrep map tensors. Each irrep map tensor holds a disjoint subset of indices of the original irrep map and a new contracted bond index.*

Proof. Let $c_1 \dots c_N$ be the coefficients defining irrep map $\boldsymbol{\mathcal{M}}$ and Z be the remainder, then define

$$m^{(A)}_{I_1 \dots I_k J} = \begin{cases} 1 & : c_1 I_1 + \cdots + c_k I_k + J \equiv Z \pmod{G} \\ 0 & : \text{otherwise} \end{cases}$$

$$\text{and} \quad m^{(B)}_{I_{k+1} \dots I_N J} = \begin{cases} 1 & : c_{k+1} I_{k+1} + \cdots + c_N I_N - J \equiv 0 \pmod{G} \\ 0 & : \text{otherwise.} \end{cases}$$

The remainder Z can alternatively be absorbed into the definition of the other factor. It then suffices to observe that contracting these tensors gives the desired result, which we can see by associating 0 or 1 with the true/false value of conditional statements defining the sparsity,

$$\begin{aligned} m_{I_1 \dots I_N} = \sum_{J=0}^{G-1} & (c_1 I_1 + \cdots + c_k I_k + J \equiv Z \bmod G) \\ & \cdot (c_{k+1} I_{k+1} + \cdots + c_N I_N - J \equiv 0 \bmod G) \\ = & \begin{cases} 1 & : c_1 I_1 + \cdots + c_N I_N \equiv Z \pmod{G} \\ 0 & : \text{otherwise.} \end{cases} \end{aligned}$$

∎

Repeated application of this Lemma implies that, for any ordering of modes, $\boldsymbol{\mathcal{M}}$ is an MPS with all ranks at most G. Consequently, any tree tensor network of this rank is a valid decomposition [31]. The irrep alignment will make use of a tree with only two nodes, which corresponds to a low-rank matrix factorization, but other factorizations may be useful.

Given any tree tensor network with factors $\boldsymbol{\mathcal{M}}^{(1)} \dots \boldsymbol{\mathcal{M}}^{(F)}$ and auxiliary (contracted) indices $J_1 \dots J_{F-1}$, we can express $\boldsymbol{\mathcal{T}}$ using a reduced form $\boldsymbol{\mathcal{R}}^{(T)}$ that contains any choice of $N-1$ indices among $I_1 \dots I_N$ and $J_1 \dots J_{F-1}$. This representation freedom is due to the presence of a one-to-one map between the irrep $I_1 \dots I_{N-1}$ and any other irrep consisting of $N-1$ indices. Such a map exists as the tensor network encodes F equations relating the $N+F-1$ variables/indices.

Modulo ordering of indices, which is immaterial, any contraction of two tensors with cyclic group symmetry can be written, for some $s, t, v \in \{0, 1, \dots\}$, as

$$w_{i_1 I_1 \dots i_s I_s j_1 J_1 \dots j_t J_t} = \sum_{k_1 K_1 \dots k_v K_v} u_{i_1 I_1 \dots i_s I_s k_1 K_1 \dots k_v K_v} v_{k_1 K_1 \dots k_v K_v j_1 J_1 \dots j_t J_t}. \tag{3.2}$$

Our tensor diagram representation of this contraction is given in Figure 10. We assume that the group symmetries assign the same coefficients to the indices $K_1 \dots K_v$ (the contraction is well-defined only if they are the same or differ by a sign, and in the latter case the sign can be absorbed into the reduced form).

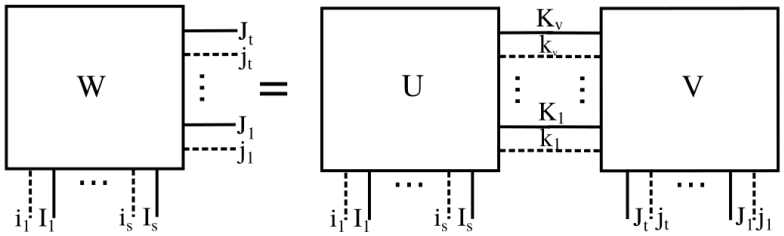

FIGURE 9. *A contraction of a tensor of order $s+v$ with a tensor of order $v+t$ into a tensor of order $s+t$, where all tensors have cyclic group symmetry and are represented with tensors of twice the order.*

The irrep alignment algorithm uses a tree with two nodes to represent each of the two input irrep maps, $\boldsymbol{\mathcal{M}}^{(U)}$ and $\boldsymbol{\mathcal{M}}^{(V)}$. Each pair of nodes separates the contracted and uncontracted indices,

$$m^{(U)}_{I_1 \dots I_s K_1 \dots K_v} = \sum_P m^{(U,I)}_{I_1 \dots I_s P} m^{(U,K)}_{K_1 \dots K_v P},$$
$$m^{(V)}_{K_1 \dots K_v J_1 \dots J_r} = \sum_Q m^{(V,K)}_{K_1 \dots K_v Q} m^{(V,J)}_{J_1 \dots J_t Q}.$$

In defining these reduced forms, we absorb the remainders associated with the irrep maps $\boldsymbol{\mathcal{M}}^{(U)}$ and $\boldsymbol{\mathcal{M}}^{(V)}$ into $\boldsymbol{\mathcal{M}}^{(U,I)}$ and $\boldsymbol{\mathcal{M}}^{(V,J)}$, respectively. Consequently, we have that $\boldsymbol{\mathcal{M}}^{(U,K)}$ and $\boldsymbol{\mathcal{M}}^{(V,K)}$ both have a remainder of zero and are both defined by the same set of coefficients, so $\boldsymbol{\mathcal{M}}^{(U,K)} = \boldsymbol{\mathcal{M}}^{(V,K)}$. Associated with these irrep maps, we define reduced forms that use a representation that includes the auxiliary indices (P and Q),

- $\boldsymbol{\mathcal{R}}^{(U)}$ with indices $I_1 \dots I_{s-1} K_1 \dots K_{v-1} P$, and
- $\boldsymbol{\mathcal{R}}^{(V)}$ with indices $K_1 \dots K_{v-1} J_1 \dots J_{t-1} Q$.

Using these irrep maps and reduced forms we can represent $\boldsymbol{\mathcal{U}}$ as

$$u_{i_1 I_1 \dots i_s I_s k_1 K_1 \dots k_v K_v} = \sum_P r^{(U)}_{i_1 I_1 \dots\dots i_{s-1} I_{s-1} i_s k_1 K_1 \dots\dots k_{v-1} K_{v-1} k_v P} m^{(U,I)}_{I_1 \dots I_s P} m^{(U,K)}_{K_1 \dots K_v P}. \tag{3.3}$$

This representation is depicted in Figure 10. With $\boldsymbol{\mathcal{V}}$ represented analogously, we can com-

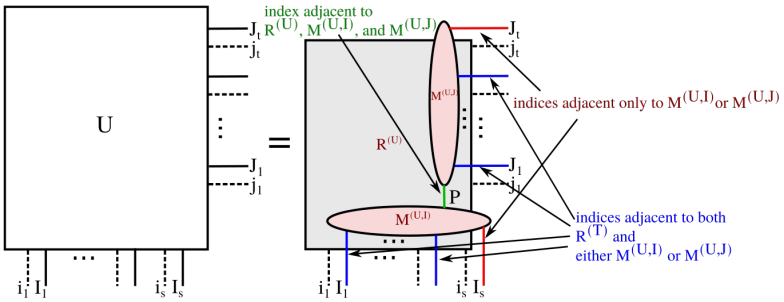

FIGURE 10. *Symmetry aligned reduced form of a group symmetric tensor. The P index belongs to the reduced form as well as both group tensor factors. The I_s and J_t indices are represented implicitly via irrep maps, i.e., are not indices of the reduced form.*

pute $\mathcal{W}$ as

$$w_{i_1 I_1 \dots i_s I_s j_1 J_1 \dots j_t J_t} = \sum_{k_1 K_1 \dots k_{v-1} K_{v-1} k_v P Q} \Bigg[r^{(U)}_{i_1 I_1 \dots i_{s-1} I_{s-1} i_s k_1 K_1 \dots k_{v-1} K_{v-1} k_v P} \, r^{(V)}_{k_1 K_1 \dots k_{t-1} K_{v-1} k_v j_1 J_1 \dots j_{t-1} J_{t-1} j_t Q} \, m^{(U,I)}_{I_1 \dots I_s P} m^{(V,J)}_{J_1 \dots J_t Q} \Big(\sum_{K_v} m^{(U,K)}_{K_1 \dots K_v P} m^{(V,K)}_{K_1 \dots K_v Q} \Big) \Bigg].$$

Consequently, since $\sum_{K_v} m^{(U,K)}_{K_1 \dots K_v P} m^{(V,K)}_{K_1 \dots K_v Q} = \delta(P, Q)$, we can eliminate the P and K_v indices and bring the remaining irrep maps outside the main summation, so

$$w_{i_1 I_1 \dots i_s I_s j_1 J_1 \dots j_t J_t} = \Big(\sum_{Q} m^{(U,I)}_{I_1 \dots I_s Q} m^{(V,J)}_{J_1 \dots J_t Q} \Big) \cdot \Bigg(\sum_{k_1 K_1 \dots k_{v-1} K_{v-1} k_v} r^{(U)}_{i_1 I_1 \dots i_{s-1} I_{s-1} i_s k_1 K_1 \dots k_{v-1} K_{v-1} k_v Q} \, r^{(V)}_{k_1 K_1 \dots k_{t-1} K_{v-1} k_v j_1 J_1 \dots j_{t-1} J_{t-1} j_t Q} \Bigg).$$

Figure 11 provides tensor diagrams depicting how the above equations permit the irrep alignment algorithm to yield an efficient contraction.

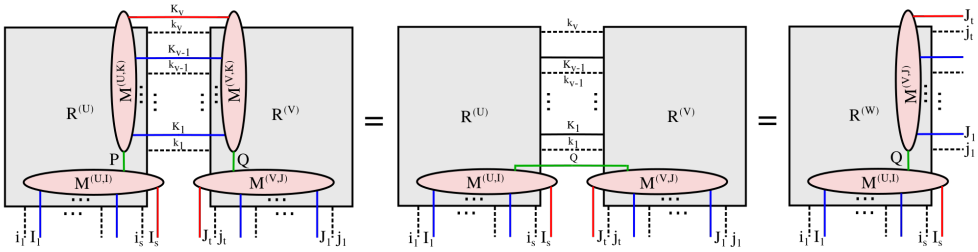

FIGURE 11. *Main contraction between aligned reduced forms. Two of the factors composing the irrep maps can be contracted away independently due to alignment of reduced forms.*

Algorithm 3.2 The irrep alignment algorithm for contraction of cyclic group symmetric tensors, for contraction defined as in (3.2).

1: Input two tensors $\boldsymbol{\mathcal{U}}$ of order $s+v$ and $\boldsymbol{\mathcal{V}}$ of order $v+t$ with group symmetries described using coefficient vectors $\boldsymbol{c}^{(U)}$ and $\boldsymbol{c}^{(V)}$ and remainders $Z^{(U)}$ and $Z^{(V)}$ as in (3.1).

2: Assume that these vectors share coefficients for contracted modes of the tensors, so that if $\boldsymbol{c}^{(U)} = \begin{bmatrix} \boldsymbol{c}_1^{(U)} \\ \boldsymbol{c}_2^{(U)} \end{bmatrix}$, then $\boldsymbol{c}^{(V)} = \begin{bmatrix} \boldsymbol{c}_2^{(U)} \\ \boldsymbol{c}_2^{(V)} \end{bmatrix}$.

3: Define new coefficient vectors, $\boldsymbol{c}^{(A)} = \begin{bmatrix} \boldsymbol{c}_1^{(U)} \\ 1 \end{bmatrix}$, $\boldsymbol{c}^{(B)} = \begin{bmatrix} \boldsymbol{c}_2^{(U)} \\ -1 \end{bmatrix}$, and $\boldsymbol{c}^{(C)} = \begin{bmatrix} \boldsymbol{c}_2^{(V)} \\ 1 \end{bmatrix}$.

4: Define irrep maps $\boldsymbol{\mathcal{M}}^{(U,I)}$, $\boldsymbol{\mathcal{M}}^{(U,K)}$, and $\boldsymbol{\mathcal{M}}^{(V,J)}$ based, respectively, based on the coefficient vectors $\boldsymbol{c}^{(A)}$, $\boldsymbol{c}^{(B)}$, $\boldsymbol{c}^{(C)}$ and remainders $Z^{(U)}$, 0, $Z^{(V)}$.

5: Obtain the irrep aligned reduced forms $\boldsymbol{\mathcal{R}}^{(U)}$ and $\boldsymbol{\mathcal{R}}^{(V)}$ for $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$, respectively, so that 3.3 is satisfied. Assuming we are given standard reduced forms $\bar{\boldsymbol{\mathcal{R}}}^{(U)}$ and $\bar{\boldsymbol{\mathcal{R}}}^{(V)}$ for $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$ that do not store the last symmetry mode (other cases are similar), we can compute $\boldsymbol{\mathcal{R}}^{(U)}$ and $\boldsymbol{\mathcal{R}}^{(V)}$ via the following contractions:

$$r^{(U)}_{i_1 I_1 \dots i_{s-1} I_{s-1} i_s k_1 K_1 \dots k_{v-1} K_{v-1} k_v Q} = \sum_{I_s K_v} \bar{r}^{(U)}_{i_1 I_1 \dots i_s I_s k_1 K_1 \dots k_{v-1} K_{v-1} k_v} m^{(U,I)}_{I_1 \dots I_s Q} m^{(U,K)}_{K_1 \dots K_v Q},$$

$$r^{(V)}_{k_1 K_1 \dots k_{v-1} K_{v-1} k_v j_1 J_1 \dots j_{t-1} J_{t-1} j_t Q} = \sum_{K_v J_t} \bar{r}^{(V)}_{k_1 K_1 \dots k_v K_v j_1 J_1 \dots j_{t-1} J_{t-1} j_t} m^{(U,K)}_{K_1 \dots K_v Q} m^{(V,J)}_{J_1 \dots J_t Q}.$$

▷ *The above contractions can be done with constant work per element of $\boldsymbol{\mathcal{R}}^{(U)}$ and $\boldsymbol{\mathcal{R}}^{(V)}$, namely $O(n^{s+v}G^{s+v-1})$ and $O(n^{v+t}G^{v+t-1})$, or with a factor of $O(G)$ more if done as dense tensor contractions that ignore the structure of $\boldsymbol{\mathcal{M}}^{(U,I)}$, $\boldsymbol{\mathcal{M}}^{(U,K)}$, and $\boldsymbol{\mathcal{M}}^{(V,J)}$.*

6: Compute

$$r^{(W)}_{i_1 I_1 \dots i_{s-1} I_{s-1} i_s J_1 J_1 \dots j_{t-1} J_{t-1} j_t Q} = \sum_{k_1 K_1 \dots k_{v-1} K_{v-1} k_v} r^{(U)}_{i_1 I_1 \dots i_{s-1} I_{s-1} i_s k_1 K_1 \dots k_{v-1} K_{v-1} k_v Q} r^{(V)}_{k_1 K_1 \dots k_{t-1} K_{v-1} k_v j_1 J_1 \dots j_{t-1} J_{t-1} j_t Q}$$

▷ *The above contraction has cost $O(n^{s+t+v}G^{s+t+v-2})$*

7: If a standard output reduced form is desired, for example with the last mode of $\boldsymbol{\mathcal{W}}$ stored implicitly, then compute

$$\bar{r}^{(W)}_{i_1 I_1 \dots i_s I_s j_1 J_1 \dots j_{t-1} J_t j_t} = \sum_Q r^{(W)}_{i_1 I_1 \dots i_{s-1} I_{s-1} i_s J_1 J_1 \dots j_{t-1} J_{t-1} j_t Q} m^{(U,I)}_{I_1 \dots I_s Q}$$

If we instead desired a reduced form with another implicit mode, it would not be implicit in $\boldsymbol{\mathcal{R}}^{(W)}$, so we would need to also contract with $m^{(V,J)}_{J_1 \dots J_t Q}$ and sum over the desired implicit mode.

▷ *In either case, the above contraction can be done with constant work per element of $\boldsymbol{\mathcal{R}}^{(W)}$, namely $O(n^{s+t}G^{s+t-1})$, or with a factor of $O(G)$ more if done as dense tensor contractions, if ignoring the structure of $\boldsymbol{\mathcal{M}}^{(U,I)}$ and $\boldsymbol{\mathcal{M}}^{(V,J)}$.*

```
import numpy as np
from symtensor import array, einsum

# Define Z3 Symmetry
irreps = [0,1,2]
G = 3
total_irrep = 0
z3sym  = ["++--", [irreps]*4, total_irrep, G]

# Initialize two sparse tensors as input
N = 10
Aarray = np.random.random([G,G,G,N,N,N,N]
Barray = np.random.random([G,G,G,N,N,N,N])

# Initialize symtensor with raw data and symmetry
u = array(Aarray, z3sym)
v = array(Barray, z3sym)

# Compute output symtensor
w = einsum('abkl,klij->abij', u,  v)
```

FIGURE 12. *Symtensor library example for contraction of two group symmetric tensors.*

4. Software Automation. We implement the irrep alignment algorithm as a Python library, Symtensor[1], that automatically maps cyclic group-symmetric tensor contractions onto a few `einsum` operations on dense tensors. The library is interfaced to different tensor contraction backends. Aside from the default NumPy `einsum`, we provide a backend that leverages MKL's batched matrix-multiplication routines [2] to obtain good threaded performance, and employ Cyclops [40] for distributed-memory execution.

The Symtensor library maps cyclic group symmetric tensors into reduced representations and automatically aligns these reduced representations so that contractions can proceed with optimal efficiency. The reduced representations are stored as dense arrays, and these objects and the resulting dense contractions are handled by the array backend. The user interface mimics the NumPy API and hides any explicit reference to symmetry except at the time of array creation. This API enables users to effectively implement numerical methods with group symmetric tensors in a symmetry-oblivious manner.

In Figure 12, we provide an example of how Symtensor library can be used to perform the contraction of two cyclic group tensors. In the code, the Symtensor library initializes the order 4 cyclic group symmetric tensor using an underlying order 7 dense reduced representation, with Z_3 (cyclic group with $G = 3$) symmetry for each index. Once the tensors are initialized, the subsequent `einsum` operation implements the contraction shown in Fig. 5 without referring to any symmetry information in its interface. While the example is based on a simple cyclic group for an order 4 tensor, the library supports arbitrary orders, as well as products of cyclic groups and infinite cyclic groups (e.g. $U(1)$ symmetries).

As introduced in Section 3, the main operations in our irrep alignment algorithm consist of transformation of the reduced form and the contraction of reduced forms. Symtensor chooses the least costly version of the irrep alignment algorithm from a space of variants defined by different choices of the three implicitly represented modes in the symmetry aligned reduced form in Algorithm 3.2 (therein these are I_s, J_t, and K_v). This choice is made by enumerating all valid variants. After choosing the best reduced form, the required irrep maps, $\boldsymbol{\mathcal{M}}^{(U,I)}$ and $\boldsymbol{\mathcal{M}}^{(V,J)}$ in Algorithm 3.2, are generated as dense tensors. This permits both the

[1] https://github.com/yangcal/symtensor

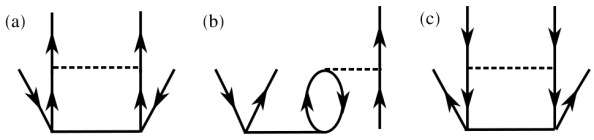

FIGURE 13. *Goldstone diagrams for the three types of CC contractions in our example. Virtual sectors are denoted with upward arrows and occupied sectors with downward arrows. Note that this is domain-specific notation that is slightly different from that used in the earlier symmetric tensor diagrams.*

transformations and the reduced form contraction to be done as `einsum` operations of dense tensors with the desired backend.

5. Example Applications. We survey a few group symmetric tensor contractions that are the costliest components of common tensor network and quantum chemistry methods. These contractions, summarized in Table 1, are evaluated as part of a benchmark suite in Section 6 (we also consider a suite of contractions with different tensor order, i.e. different choices of s, t, v).

5.1. Periodic Coupled Cluster Contractions. We investigate the performance of the irrep alignment contraction algorithm for several expensive tensor contractions arising in the doubles amplitude equation of the CC theory of periodic systems. These can be written as

$$w_{aA,bB,iI,jJ} = \sum_{cC,dD} u_{aA,bB,cC,dD} v_{iI,jJ,cC,dD},$$
$$x_{aA,bB,iI,cC} = \sum_{kK,lL} u_{kK,lL,aA,bB} v_{kK,lL,cC,iI},$$
$$y_{aA,bB,iI,jJ} = \sum_{kK,lL} u_{kK,lL,iI,jJ} v_{kK,lL,aA,bB}.$$

These contractions are displayed in Fig.13 as diagrams in a form that is common in quantum chemistry literature. As discussed in Section 2, in crystalline (periodic) materials, the crystal translational group is the product of cyclic groups associated with the lattice vectors of the crystal. In 3 dimensions, the order of group, known as the total number of k points, takes the form $G = G_1 \times G_2 \times G_3$, where G_1, G_2, G_3 are the number of k points along each dimension. Each index of the tensors is associated with this group symmetry. The indices also fall into two classes, i, j, k, l (the occupied indices) and a, b, c, d (the virtual indices) associated with different dimensions $N_{\text{occ}} = N_i = N_j = N_k = N_l$ and $N_{\text{virt}} = N_a = N_b = N_c = N_d$. The cost of the above contractions using the irrep alignment algorithm scales as $G^4 N_{occ}^2 N_{vir}^4$, $G^4 N_{occ}^3 N_{vir}^3$ and $G^4 N_{occ}^4 N_{vir}^2$ respectively. These contractions are summarized in Table 1 where, for simplicity, we use $N = N_{vir} = N_{occ}$.

5.2. Tensor Network Contractions. We benchmark the performance of the irrep alignment algorithm for two tensor contractions arising in tensor network simulations, one arising in DMRG calculations with MPS, and the other in a 2D PEPS contraction. The tensor contractions are illustrated shown in Fig. 14.

The DMRG contraction considered is the absorption of the blocked environment tensor $\boldsymbol{\mathcal{U}}$ into the MPS tensor $\boldsymbol{\mathcal{V}}$,

$$w_{iI,jJ,lL,mM} = \sum_{kK} u_{iI,jJ,kK} v_{kK,lL,mM},$$

which is encountered while solving the local eigenvalue problem. Here, as illustrated in Fig. 14a, two environment tensors are contracted with a local MPS and MPO tensor; we

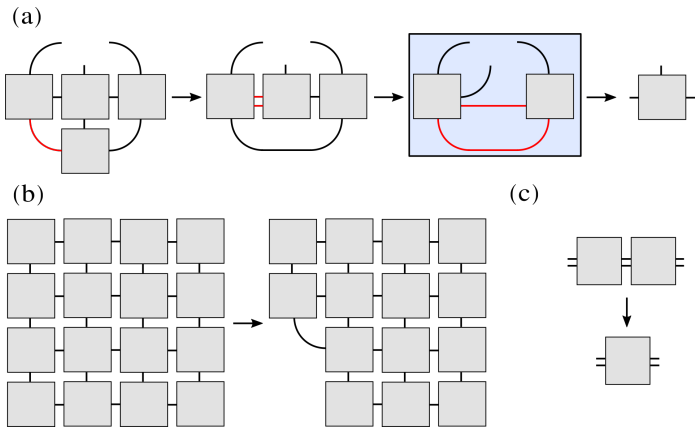

FIGURE 14. *Tensor network diagrams for the (a) DMRG and (b-c) PEPS contractions considered. In (a), two environment tensors are contracted with an MPS and an MPO tensor in three steps, with the contraction considered highlighted in blue. For the example PEPS contraction, (b) illustrates how a single layer PEPS is contracted column by column using the IBMPS algorithm and (c) shows the diagram for the considered contraction, which is part of the randomized implicit SVD step.*

benchmark the third contraction, highlighted in blue. While in general the number of symmetry blocks is different for each of the tensor indices in a tensor network, for simplicity we keep the number of symmetry blocks constant for all legs, meaning $G = G_I = G_J = G_K = G_L = G_M$. The indices i, k, and m denote the auxiliary indices of the MPS with sizes $N_{\text{mps}} = N_i = N_k = N_m$, j is the auxiliary index of the MPO used for the eigenvalue problem with size N_j, and l is the physical bond dimension of size N_l.

The PEPS contraction considered is one that arises in a recently proposed variant of the boundary contraction algorithm [48], the Implicit Boundary MPS (IBMPS) [33], which provides asymptotic speedup over the standard Boundary MPS (BMPS) method [23] for a given bond dimension by using an implicit randomized singular value decomposition (SVD) in the boundary bond dimension truncation. The IBMPS procedure approximately contracts a square PEPS lattice one layer at a time, with Fig. 14b showing an example of this; we benchmark one of the contractions with the highest scaling with the PEPS bond dimension, shown in Fig. 14c. This contraction is of tensors $\boldsymbol{\mathcal{U}}$ and $\boldsymbol{\mathcal{V}}$,

$$w_{iI,jJ,mM,nN} = \sum_{kK,lL} u_{iI,jJ,kK,lL} v_{kK,lL,mM,nN}.$$

Indices i and j have block sizes $N_{\text{mps}} = N_i = N_j$, corresponding to the auxiliary bond dimension of the boundary MPS, and the remaining indices correspond to the PEPS auxiliary bond dimension, with block sizes $N_{\text{peps}} = N_k = N_l = N_m = N_n$. All indices have the same number of symmetry blocks, G. The costs of these DMRG and PEPS contractions using the irrep alignment algorithm scale as $G^3 N_{\text{mps}}^3 N_j N_l$ and $G^4 N_{\text{mps}}^2 N_{\text{peps}}^4$ respectively; in Table 1 we set $N = N_{\text{mps}} = N_{\text{peps}}$ for simplicity.

6. Performance Evaluation. Performance experiments were carried out on the Stampede2 supercomputer. Each Stampede2 node is a Intel Knight's Landing (KNL) processor, on which we used up to 64 of 68 cores by employing up to 64 threads with single-node NumPy/MKL and 64 MPI processes per node with 1 thread per process with Cyclops. We used the Symtensor library together with one of three external contraction backends: Cyclops, default NumPy, or a batched BLAS backend for NumPy arrays (this backend leverages

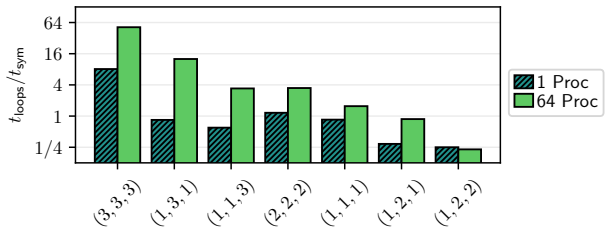

FIGURE 15. *Comparison of execution times for various types of contraction using the Symtensor Library with batched BLAS and using loops over blocks with NumPy as the contraction backend. Results are shown for various combinations of (s, t, v) where the number of indices shared between the first input and output is s, between the second input and output is t, and between the two inputs (contracted) is v.*

HPTT [41] for fast tensor transposition and dispatches to Intel's MKL BLAS for batched matrix multiplication).[2] We also compare against the loop over symmetry blocks algorithm as illustrated in Algorithm 3.1.

6.1. Single-Node Performance Results. We consider the performance of Symtensor on a single core and a single node of KNL relative to manually-implemented looping over blocks as well as relative to naive contractions that ignore symmetry. Our manual loop implementations of contractions store a Python list of NumPy arrays to represent the tensor blocks and invokes the NumPy `einsum` functions to perform each blockwise contraction.

6.1.1. Sensitivity to Contraction Type. We first examine the performance of the irrep alignment algorithm for generic contractions (Eq. 3.2) with equal dimensions N and G for each mode, but different (s, t, v). Figure 15 shows the speed-up in execution time obtained by Symtensor relative to the manual loop implementation. The contractions are constructed by fixing $G = 4$ and modifying N to attain a fixed total of 80×10^9 floating-point operations. All contractions are performed on both a single thread and 64 threads of a single KNL node and timings are compared in those respective configurations. The irrep alignment algorithm achieves better parallel scalability than block-wise contraction and can also be faster sequentially. However, we also observe that in cases when one tensor is larger than others (when s, t, v are unequal) the irrep alignment approach can incur overhead relative to the manual loop implementation. Overhead can largely be attributed to the cost of transformations between reducible representations, which are done as dense tensor contractions. An alternative transformation mechanism that forgoes this factor of $O(G)$ overhead would likely reduce sequential efficiency for these cases, but the use of dense tensor contractions permits use of existing optimized kernels and easy parallelizability.

6.1.2. Sensitivity to Symmetry Group Size for Application-Specific Contractions. The results are displayed in Figure 16 with the left, center, and right plots showing the scaling for the contractions labeled MM, CC_1, and PEPS in Table 1. We compare scaling relative to two conventional approaches: a dense contraction without utilizing symmetry and loops over symmetry blocks, both using NumPy's `einsum` function. The sizes of the tensors considered are, for matrix multiplication, $N = 500$ and $G \in [4, 12]$, for the CC contraction, $N_{occ} = 8$, $N_{vir} = 16$, with $G \in [4, 12]$, and for the PEPS contraction, $N_{\text{mps}} = 16$, $N_{\text{peps}} = 4$, with $G \in [2, 10]$. For all but the smallest contractions, using the Symtensor implementation improves contraction performance. A comparison of the slopes of the lines in each of the

[2]We used the default Intel compilers on Stampede2 with the following software versions: HPTT v1.0.0, CTF v1.5.5 (compiled with optimization flags: -O2 -no-ipo), and MKL v2018.0.2.

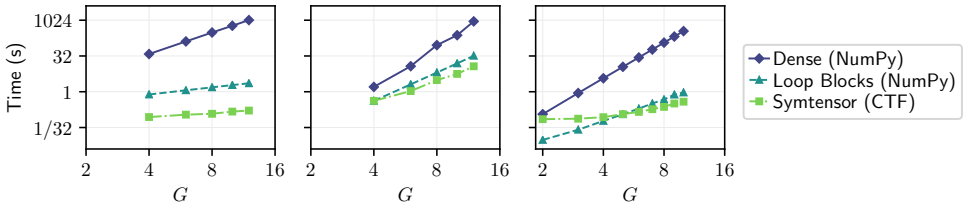

FIGURE 16. *Comparison of the execution times, in seconds, for contractions on a single thread using three different algorithms, namely a dense, non-symmetric contraction, loops over symmetry blocks, and our Symtensor library. From left to right, the plots show the scaling for matrix multiplication (MM), a coupled cluster contraction (CC_1), and a tensor network contraction (PEPS). The dense and loop over blocks calculations use NumPy as a contraction backend, while the Symtensor library here uses Cyclops as the contraction backend.*

TABLE 1

Summary of coupled cluster and tensor network contractions used to benchmark the symmetric tensor contraction scheme and their costs. We include matrix multiplication (MM) as a point of reference.

Label	Contraction	Symmetric Cost	Naive Cost
MM	$w_{iI,kK} = \sum_{jJ} u_{iI,jJ} v_{jJ,kK}$	$\mathcal{O}(GN^3)$	$\mathcal{O}(G^3N^3)$
CC_1	$w_{aA,bB,iI,jJ} = \sum_{cC,dD} u_{aA,bB,cC,dD} v_{iI,jJ,cC,dD}$	$\mathcal{O}(G^4N^6)$	$\mathcal{O}(G^6N^6)$
CC_2	$x_{aA,bB,iI,cC} = \sum_{kK,lL} u_{kK,lL,aA,bB} v_{kK,lL,cC,iI}$	$\mathcal{O}(G^4N^6)$	$\mathcal{O}(G^6N^6)$
CC_3	$y_{aA,bB,iI,jJ} = \sum_{kK,lL} u_{kK,lL,iI,jJ} v_{kK,lL,aA,bB}$	$\mathcal{O}(G^4N^6)$	$\mathcal{O}(G^6N^6)$
MPS	$w_{iI,jJ,lL,mM} = \sum_{kK} u_{iI,jJ,kK} v_{kK,lL,mM}$	$\mathcal{O}(G^3N^5)$	$\mathcal{O}(G^5N^5)$
PEPS	$w_{iI,jJ,mM,nN} = \sum_{kK,lL} u_{iI,jJ,kK,lL} v_{kK,lL,mM,nN}$	$\mathcal{O}(G^4N^6)$	$\mathcal{O}(G^6N^6)$

three plots demonstrates that the dense tensor contraction scheme results in a higher order asymptotic scaling of cost in G than either of the symmetric approaches.

Figure 17 provides absolute performance with 1 thread and 64 threads for all contractions in Table 1. For each contraction, we consider one with a large number of symmetry sectors (G) with small block size (N) (labeled with a subscript a) and another with fewer symmetry sectors and larger block size (labeled with a subscript b). The specific sizes of all tensors studied are provided in Table 2. For each of these cases, we compare the execution time, in seconds, using loops over blocks dispatching to NumPy contractions, the Symtensor library with NumPy arrays and batched BLAS as the contraction backend, and the Symtensor library using Cyclops as the array and contraction backend.

A clear advantage in parallelizability of Symtensor is evident in Figure 17. With 64 threads, Symtensor outperforms manual looping by a factor of at least 1.4X for all contraction benchmarks, and the largest speed-up, 69X, is obtained for the CC_{3a} contraction. There is a significant difference between the contractions labeled to be of type a (large G and small N) and type b (large N and small G), with the geometric mean speedup for these two being 11X and 2.8X respectively on 64 threads; on a single thread, this difference is again observed, although less drastically, with respective geometric mean speedups of 1.9X and 1.2X. Type b cases involve more symmetry blocks, amplifying overhead of manual looping.

6.2. Multi-Node Performance Results. We now illustrate the parallelizability of the irrep alignment algorithm by studying scalability across multiple nodes with distributed memory. All parallelization in Symtensor is handled via the Cyclops library in this case. The solid lines in Figure 18 show the strong scaling (fixed problem size) behavior of the Symtensor implementation on up to eight nodes of Stampede2. As a reference, we provide comparison to

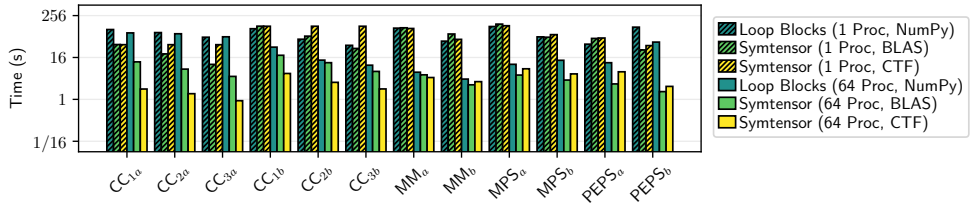

FIGURE 17. *Comparison of contraction times using the Symtensor library (using Cyclops for the array storage and contraction backened, or NumPy as the array storage with batched BLAS contraction backend) and loops over blocks using NumPy as the contraction backend. Results are shown for instances of the prototypical contractions introduced in Section 5, with details of tensor dimensions provided in Table 2. The different bars indicate both the algorithm and backend used and the number of threads used on a single node.*

TABLE 2
Dimensions of the tensors used for contractions in Figure 17 and Figure 18.

Label	Specifications
CC_{1a}	$G = 8, N_i = N_j = 16, N_a = N_b = N_c = N_d = 32$
CC_{2a}	$G = 8, N_i = N_k = N_l = 16, N_a = N_b = N_c = 32$
CC_{3a}	$G = 8, N_i = N_j = N_k = N_l = 16, N_a = N_b = 32$
CC_{1b}	$G = 16, N_i = N_j = 8, N_a = N_b = N_c = N_d = 16$
CC_{2b}	$G = 16, N_i = N_k = N_l = 8, N_a = N_b = N_c = 16$
CC_{3b}	$G = 16, N_i = N_j = N_k = N_l = 8, N_a = N_b = 16$
MM_a	$G = 2, N = 10000$
MM_b	$G = 100, N = 2000$
MPS_a	$G = 2, N_i = N_k = N_m = 3000, N_j = 10, N_l = 1$
MPS_b	$G = 5, N_i = N_k = N_m = 700, N_j = 10, N_l = 1$
PEPS_a	$G = 2, N_i = N_j = 400, N_k = N_l = N_m = N_n = 20$
PEPS_b	$G = 10, N_i = N_j = 64, N_k = N_l = N_m = N_n = 8$

strong scaling on a single node for the loop over blocks method using NumPy as the array and contraction backend. We again observe that the Symtensor irrep alignment implementation provides a significant speedup over the loop over blocks strategy, which is especially evident when there are many symmetry sectors in each tensor. For example, using 64 threads on a single node, the speedup achieved by Symtensor over the loop over blocks implementation is 41X for CC_{1a}, 5.7X for CC_{1b}, 4.1X for PEPS_a and 27X for PEPS_b. We additionally see that the contraction times continue to scale with good efficiency when the contraction is spread across multiple nodes.

Finally, in Figure 19 we display weak scaling performance, where the dimensions of each tensor are scaled with the number of nodes (starting with the problem size reported in Table 2 on 1 node) used so as to fix the tensor size per node. Thus, in this experiment, we utilize all available memory and seek to maximize performance rate. Figure 19 displays the performance rate per node, which varies somewhat across contractions and node counts, but generally does not fall off with increasing node count, demonstrating good weak scalability. When using 4096 cores, the overall performance rate approaches 4 Teraflops/s for some contractions, but is lower in other contractions that have less arithmetic intensity.

7. Conclusion. The irrep alignment algorithm leverages conservation laws implicit in cyclic group symmetry to provide a contraction method that is efficient accross a wide range

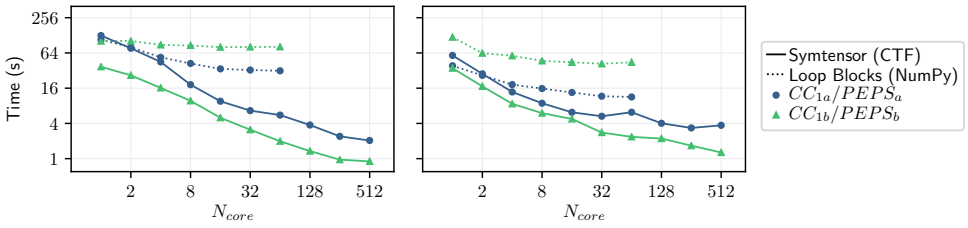

FIGURE 18. *Strong scaling across up to 8 nodes for the CC contractions (left) labelled CC_{1a} (blue circles) and CC_{1b} (green triangles) and the PEPS contractions labelled $PEPS_a$ (blue circles) and $PEPS_b$ (green triangles). The dashed lines correspond to calculations done using a loop over blocks algorithm with a NumPy contraction backend while the solid lines correspond to Symtensor calculations using the irrep alignment algorithm, with a Cyclops contraction backend.*

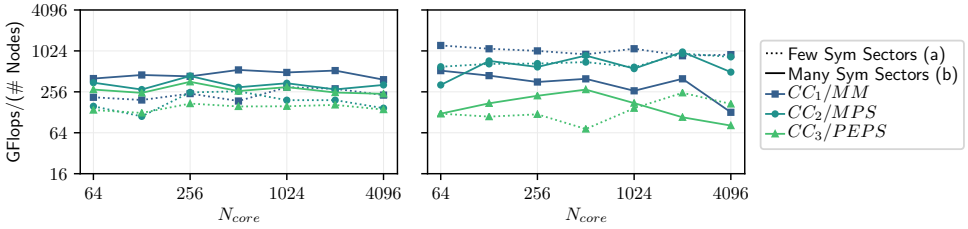

FIGURE 19. *Weak scaling (see text for details) across up to 64 nodes for CC (left) and TN (right) contractions, showing the performance, in GFlops per node, as a function of the number of used nodes. The dashed lines correspond to contractions with few symmetry irreps, previously labelled (a), while solid lines correspond to contractions with many symmetry irreps, labelled (b). The blue squares correspond to the CC_1 and matrix multiplication performance, the dark green circles correspond to the CC_2 and MPS performance, and the light green triangles correspond to the CC_3 and PEPS performance.*

of tensor contractions. This technique is applicable to many numerical methods for quantum-level modelling of physical systems that involve tensor contractions and tensor networks. The automatic handling of group symmetry with dense tensor contractions provided via the Symtensor library provides benefits in productivity, portability, and parallel scalability for such applications.

8. Acknowledgement. We thank Linjian Ma for providing the batched BLAS backend used in our calculations. ES was supported by the US NSF OAC SSI program, via awards No. 1931258 and No. 1931328. YG, PH, GKC were supported by the US NSF OAC SSI program, award No. 1931258. PH was also supported by a NSF Graduate Research Fellowship via grant DGE-1745301 and an ARCS Foundation Award. The work made use of the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by US National Science Foundation grant number ACI-1548562. We use XSEDE to employ Stampede2 at the Texas Advanced Computing Center (TACC) through allocation TG-CCR180006.

REFERENCES

[1] ITensor Library (version 2.0.11) http://itensor.org.

[2] A. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, *Performance, design, and autotuning of batched GEMM for GPUs*, in International Conference on High Performance Computing, Springer, 2016, pp. 21–38.
[3] R. J. Bartlett and M. Musiał, *Coupled-cluster theory in quantum chemistry*, 79 (2007), pp. 291–352.
[4] G. K.-L. Chan and M. Head-Gordon, *Highly correlated calculations with a polynomial cost algorithm: A study of the density matrix renormalization group*, The Journal of chemical physics, 116 (2002), pp. 4462–4476.
[5] G. K.-L. Chan, A. Keselman, N. Nakatani, Z. Li, and S. R. White, *Matrix product operators, matrix product states, and ab initio density matrix renormalization group algorithms*, The Journal of chemical physics, 145 (2016), p. 014102.
[6] F. A. Cotton, *Chemical applications of group theory*, John Wiley & Sons, 2003.
[7] T. D. Crawford and H. F. Schaefer, *An introduction to coupled cluster theory for computational chemists*, vol. 14, VCH Publishers, New York, 2000, ch. 2, pp. 33–136.
[8] G. M. Crosswhite, A. C. Doherty, and G. Vidal, *Applying matrix product operators to model systems with long-range interactions*, Physical Review B, 78 (2008), p. 035116.
[9] M. Fannes, B. Nachtergaele, and R. F. Werner, *Finitely correlated states on quantum spin chains*, Communications in mathematical physics, 144 (1992), pp. 443–490.
[10] A. L. Fetter and J. D. Walecka, *Quantum theory of many-particle systems*, Courier Corporation, 2012.
[11] N. Flocke and R. Bartlett, *Correlation energy estimates in periodic extended systems using the localized natural bond orbital coupled cluster approach*, The Journal of chemical physics, 118 (2003), pp. 5326–5334.
[12] J. Gauss, *Coupled-cluster Theory*, John Wiley & Sons, Ltd, 2002, https://doi.org/10.1002/0470845015.cca058.
[13] S. Hirata, *Tensor Contraction Engine: Abstraction and automated parallel implementation of configuration-interaction, coupled-cluster, and many-body perturbation theories*, The Journal of Physical Chemistry A, 107 (2003), pp. 9887–9897, https://doi.org/10.1021/jp034596z.
[14] S. Hirata, R. Podeszwa, M. Tobita, and R. J. Bartlett, *Coupled-cluster singles and doubles for extended systems*, The Journal of chemical physics, 120 (2004), pp. 2581–2592.
[15] F. Hummel, T. Tsatsoulis, and A. Grüneis, *Low rank factorization of the Coulomb integrals for periodic coupled cluster theory*, The Journal of chemical physics, 146 (2017), p. 124105.
[16] K. Z. Ibrahim, S. W. Williams, E. Epifanovsky, and A. I. Krylov, *Analysis and tuning of libtensor framework on multicore architectures*, in 2014 21st International Conference on High Performance Computing (HiPC), IEEE, 2014, pp. 1–10.
[17] M. Kállay and P. R. Surján, *Higher excitations in coupled-cluster theory*, 115 (2001), pp. 2945–2954, https://doi.org/10.1063/1.1383290.
[18] Y.-J. Kao, Y.-D. Hsieh, and P. Chen, *Uni10: An open-source library for tensor network algorithms*, in Journal of Physics: Conference Series, vol. 640, IOP Publishing, 2015, p. 012040.
[19] R. A. Kendall, E. Apra, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha, T. Straatsma, T. L. Windus, and A. T. Wong, *High performance computational chemistry: An overview of NWChem a distributed parallel application*, Computer Physics Communications, 128 (2000), pp. 260 – 283.
[20] Y. Kurashige and T. Yanai, *High-performance ab initio density matrix renormalization group method: Applicability to large-scale multireference problems for metal compounds*, The Journal of chemical physics, 130 (2009), p. 234114.
[21] P.-W. Lai, K. Stock, S. Rajbhandari, S. Krishnamoorthy, and P. Sadayappan, *A framework for load balancing of tensor contraction expressions via dynamic task partitioning*, in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013, pp. 1–10.
[22] M. Lax, *Symmetry principles in solid state and molecular physics*, Courier Corporation, 2001.
[23] M. Lubasch, J. I. Cirac, and M.-C. Bañuls, *Algorithms for finite projected entangled pair states*, Physical Review B, 90 (2014), https://doi.org/10.1103/physrevb.90.064425, http://dx.doi.org/10.1103/PhysRevB.90.064425.
[24] D. A. Matthews, *On extending and optimising the direct product decomposition*, Molecular Physics, 117 (2019), pp. 1325–1333.
[25] J. McClain, Q. Sun, G. K.-L. Chan, and T. C. Berkelbach, *Gaussian-based coupled-cluster theory for the ground-state and band structure of solids*, Journal of chemical theory and computation, 13 (2017), pp. 1209–1218.
[26] I. P. McCulloch and M. Gulácsi, *The non-abelian density matrix renormalization group algorithm*, EPL (Europhysics Letters), 57 (2002), p. 852.
[27] M. Motta, S. Zhang, and G. K.-L. Chan, *Hamiltonian symmetries in auxiliary-field quantum Monte Carlo calculations for electronic structure*, Physical Review B, 100 (2019), p. 045127.
[28] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, *Global Arrays: A nonuniform memory ac-*

cess programming model for high-performance computers, The Journal of Supercomputing, 10 (1996), pp. 169–189.

[29] J. NOGA AND R. BARTLETT, *The full CCSDT model for molecular electronic structure*, 86 (1987), pp. 7041–7050, https://doi.org/10.1063/1.452353.

[30] R. OLIVARES-AMAYA, W. HU, N. NAKATANI, S. SHARMA, J. YANG, AND G. K.-L. CHAN, *The ab-initio density matrix renormalization group in practice*, The Journal of chemical physics, 142 (2015), p. 034102.

[31] I. V. OSELEDETS, *Tensor-train decomposition*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2295–2317.

[32] D. OZOG, J. R. HAMMOND, J. DINAN, P. BALAJI, S. SHENDE, AND A. MALONY, *Inspector-executor load balancing algorithms for block-sparse tensor contractions*, in 2013 42nd International Conference on Parallel Processing, IEEE, 2013, pp. 30–39.

[33] Y. PANG, T. HAO, A. DUGAD, Y. ZHOU, AND E. SOLOMONIK, *Efficient 2D tensor network simulation of quantum systems*, 2020, https://arxiv.org/abs/2006.15234.

[34] C. PENG, J. A. CALVIN, F. PAVOSEVIC, J. ZHANG, AND E. F. VALEEV, *Massively parallel implementation of explicitly correlated coupled-cluster singles and doubles using TiledArray framework*, The Journal of Physical Chemistry A, 120 (2016), pp. 10231–10244.

[35] G. D. PURVIS III AND R. J. BARTLETT, *A full coupled-cluster singles and doubles model: The inclusion of disconnected triples*, The Journal of Chemical Physics, 76 (1982), pp. 1910–1918, https://doi.org/10.1063/1.443164, http://link.aip.org/link/?JCP/76/1910/1.

[36] P. SCHMOLL AND R. ORUS, *Benchmarking global $SU(2)$ symmetry in 2d tensor network algorithms*, arXiv preprint arXiv:2005.02748, (2020).

[37] U. SCHOLLWÖCK, *The density-matrix renormalization group in the age of matrix product states*, Annals of physics, 326 (2011), pp. 96–192.

[38] G. E. SCUSERIA, A. C. SCHEINER, T. J. LEE, J. E. RICE, AND H. F. SCHAEFER III, *The closed-shell coupled cluster single and double excitation (CCSD) model for the description of electron correlation. A comparison with configuration interaction (CISD) results*, The Journal of chemical physics, 86 (1987), pp. 2881–2890.

[39] S. SHARMA AND G. K.-L. CHAN, *Spin-adapted density matrix renormalization group algorithms for quantum chemistry*, The Journal of chemical physics, 136 (2012), p. 124121.

[40] E. SOLOMONIK, D. MATTHEWS, J. R. HAMMOND, J. F. STANTON, AND J. DEMMEL, *A massively parallel tensor contraction framework for coupled-cluster computations*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3176–3190.

[41] P. SPRINGER, T. SU, AND P. BIENTINESI, *HPTT: A High-Performance Tensor Transposition C++ Library*, in Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming, ARRAY 2017, New York, NY, USA, 2017, ACM, pp. 56–62, https://doi.org/10.1145/3091966.3091968, http://doi.acm.org/10.1145/3091966.3091968.

[42] J. F. STANTON, *Why CCSD(T) works: a different perspective*, 281 (1997), pp. 130–134.

[43] J. F. STANTON, J. GAUSS, J. D. WATTS, AND R. J. BARTLETT, *A direct product decomposition approach for symmetry exploitation in many-body methods. I. Energy calculations*, The Journal of Chemical Physics, 94 (1991), pp. 4334–4345.

[44] S. STERNBERG, *Group theory and physics*, Cambridge University Press, 1995.

[45] A. SZABO AND N. S. OSTLUND, *Modern quantum chemistry: introduction to advanced electronic structure theory*, Courier Corporation, 2012.

[46] J. ČÍŽEK, *On the correlation problem in atomic and molecular systems. Calculation of wavefunction components in Ursell-type expansion using quantum-field theoretical methods*, The Journal of Chemical Physics, 45 (1966), pp. 4256–4266.

[47] J. ČÍŽEK AND J. PALDUS, *Correlation problems in atomic and molecular systems III. rederivation of the coupled-pair many-electron theory using the traditional quantum chemical methods*, International Journal of Quantum Chemistry, 5 (1971), pp. 359–379, https://doi.org/10.1002/qua.560050402, http://dx.doi.org/10.1002/qua.560050402.

[48] F. VERSTRAETE AND J. I. CIRAC, *Renormalization algorithms for quantum-many body systems in two and higher dimensions*, (2004), https://arxiv.org/abs/cond-mat/0407066.

[49] F. VERSTRAETE, J. J. GARCIA-RIPOLL, AND J. I. CIRAC, *Matrix product density operators: simulation of finite-temperature and dissipative systems*, Physical review letters, 93 (2004), p. 207204.

[50] G. VIDAL, *Class of quantum many-body states that can be efficiently simulated*, Physical review letters, 101 (2008), p. 110501.

[51] S. R. WHITE, *Density matrix formulation for quantum renormalization groups*, Physical review letters, 69 (1992), p. 2863.

[52] S. R. WHITE, *Density-matrix algorithms for quantum renormalization groups*, Physical Review B, 48 (1993), p. 10345.

[53] K. YE AND L.-H. LIM, *Tensor network ranks*, arXiv preprint arXiv:1801.02662, (2018).